PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Gabriel Felipe Pinheiro Couto

ANÁLISE PREDITIVA MEDIANTE A MACHINE LEARNING:
Estudo com dados do Titanic

São Paulo
2022

Gabriel Felipe Pinheiro Couto

ANÁLISE PREDITIVA MEDIANTE A MACHINE LEARNING:
Estudo com dados do Titanic

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

São Paulo
2022

SUMÁRIO

1. Introdução

Todo mundo conhece essa linda história de amor: Jack e Rose, um amor impossível em um navio romântico que ultrapassa as amarras de classes sociais e preconceitos. Mas o maior e mais seguro transatlântico do mundo, que demorou longos 4 anos para ser construído, naufragou tragicamente no dia 14 de abril de 1912. O naufrágio do RMS Titanic é um dos naufrágios mais infames da história. Em 15 de abril de 1912, durante sua viagem inaugural, o Titanic afundou após colidir com um iceberg, matando 1502 de 2224 passageiros e tripulação. Esta sensacional tragédia chocou a comunidade internacional e levou a melhores regulamentações de segurança para os navios.

1.1. Contextualização

O presente estudo tem por finalidade uma análise sobre os dados da história verídica do navio chamado RMS Titanic, o qual foi construído em Belfast na Irlanda do Norte, assim como, a viagem inaugural e o triste desfecho do navio ao se chocar com o iceberg. Analisa-se, que o Titanic foi idealizado, sonhado e projetado pelos melhores projetistas, trabalhadores e operários da época, para ser o maior navio do mundo, o mais luxuoso e insubmersível. A sofisticação e a grandeza do Titanic eram surpreendentes, o qual fascinou todos em sua primeira e única viagem.

O navio foi o maior já existente, sendo o maior objeto removível criado até agora, segundo Geoff Tibbals relatou em seu livro “The Titanic”. A grande história do "navio inafundável" - o navio tinha 269 metros e pesava 46.000 toneladas. A embarcação organizou cerca de 2345 passageiros e 860 tripulantes, estava equipado apenas com os melhores itens associados à sensação de classe alta, como se essas pessoas estivessem em uma hospedagem ainda mais luxuosa do que aquelas que estavam em terra. Era considerado o navio mais seguro já feito. Em 14 de abril de 1912 o Titanic começa a afundar. Sinais de socorro foram enviados, mas quase nunca recebidos, porque não havia comunicação ao redor, havia apenas cerca de 20 barcos salva-vidas atendendo apenas metade das pessoas a bordo (2200). A

prioridade dos seus barcos salva-vidas eram mulheres e crianças, nesse caso, primeira classe e depois classe baixa.

Nesse contexto, mergulhamos nos dados de sobreviventes e pessoas que perderam suas vidas, analisando qual foi a probabilidade de sobrevivência delas. Uma das razões para que este naufrágio seja considerado tão trágico foi a falta de barcos salva-vidas para todas as pessoas a bordo. Embora houvesse algum elemento de sorte envolvido em sobreviver ao afundamento, alguns grupos de pessoas tinham maior probabilidade de sobreviver do que outros, como mulheres, crianças e a classe alta.

1.2. O problema proposto

Às 00h05 de 15 de abril, o Capitão Smith ordenou que os botes salva-vidas fossem descobertos, que os passageiros fossem reunidos e se preparassem. Também ordenou aos operadores de rádio que começassem a enviar pedidos de socorro, pedidos esses que colocaram o navio no lado oeste do cinturão de gelo e direcionaram os barcos que vieram em socorro para uma posição que resultou ser equivocada em cerca de 13,5 milhas náuticas (15,5 mi / 25 km).

Abaixo dos decks, a água estava entrando nos níveis mais baixos do navio. À medida que a sala de correio inundava, os classificadores do correio faziam uma última e inútil tentativa de salvar os 400 mil itens carregados a bordo do Titanic. Em outros lugares, já era possível ouvir o ar sendo forçado para fora pela água que entrava. Acima deles, os camareiros foram de porta em porta, despertando passageiros e tripulação – o Titanic não tinha um sistema de aviso por alto-falantes – e pedindo que se dirigissem ao Convés dos Botes.

O *Titanic* tinha um total de 20 botes salva-vidas, composto por 16 botes de madeira nos turcos, 8 em cada lado do navio, e 4 desmontáveis com fundos de madeira e laterais de lona. Os desmontáveis estavam guardados de cabeça para baixo com os lados dobrados, e teriam que ser erguidos e movidos pelos turcos para o lançamento. Dois foram guardados sob os barcos de madeira e os outros dois foram amarrados no topo dos quartos dos oficiais. A posição destes últimos os tornaria extremamente difíceis de lançar, pois pesavam várias toneladas cada um e tinham que ser manipulados até o convés do navio.

Em média, os botes salva-vidas podiam levar até 68 pessoas cada, e coletivamente podiam acomodar 1 178 – quase metade do número de pessoas a bordo e um terço do número que o navio estava licenciado para transportar. A falta de botes salva-vidas não foi por falta de espaço nem por causa do custo. O *Titanic* foi projetado para acomodar até 68 botes salva-vidas suficiente para todos a bordo, e o preço para 32 botes extras custam por volta de apenas 16 mil dólares (equivalente à cerca de 397 mil dólares em 2016), uma pequena fração dos 7,5 milhões de dólares que a empresa gastou no *Titanic*. Em uma emergência, os botes salva-vidas seriam destinados para transferir passageiros do navio afligido para um navio próximo. Portanto, era comum que os navios de passageiros tivessem muito menos botes salva-vidas do que os necessários para acomodar todos os seus passageiros e tripulantes, e dos 39 navios de passageiros britânicos da época com mais de 10 mil toneladas, 33 tinham poucos lugares nos botes salva-vidas para acomodar todos a bordo. A White Star Line desejava que o navio tivesse um amplo deck de passeio com vistas ininterruptas do mar, o que teria sido obstruído por uma fileira contínua de botes salva-vidas.

O número exato de mortos no naufrágio é incerto devido a vários fatores, como a confusão sobre a lista de passageiros, que incluía nomes de pessoas que cancelaram a viagem no último momento e o fato de alguns passageiros terem embarcado sob pseudônimos por várias razões e foram contados em duas listas de vítimas. O número total de mortos já foi colocado entre 1 490 e 1 635 pessoas. Menos de um terço daqueles que estavam a bordo do *Titanic* sobreviveram. Alguns sobreviventes morreram pouco tempo depois; ferimentos e os efeitos da exposição ao frio causaram a morte de muitos daqueles resgatados pelo *Carpathia*. 49% das crianças, 26% das mulheres, 82% dos homens e 78% da tripulação morreram. Os números mostram enormes diferenças nos índices de sobrevivência das diferentes classes a bordo do navio, especialmente entre as mulheres e crianças. Apesar de menos de 10% das mulheres da primeira e segunda classe juntas terem morrido, 54% daquelas na terceira pereceram. Similarmente, cinco das seis crianças na primeira classe e todas da segunda classe sobreviveram, porém 52 das 79 na terceira morreram. A única criança da primeira classe a morrer foi Loraine Allison, de dois anos de idade. Proporcionalmente, as maiores perdas foram sofridas por homens da segunda classe, dos quais 92% morreram. Além

disso, três animais de estimação que estavam a bordo sobreviveram. Os números abaixo são do relatório do inquérito britânico sobre o desastre.

Passageiros	Categoria	Número a bordo	Porcentagem por total a bordo	Salvos	Mortos	Porcentagem salvos	Porcentagem mortos	Porcentagem salvos por total a bordo	Porcentagem mortos por total a bordo
Crianças	1ª Classe	6	0,3%	5	1	83%	17%	0,2%	0,04%
	2ª Classe	24	1,1%	24	0	100%	0%	1,1%	0%
	3ª Classe	79	3,6%	27	52	34%	66%	1,2%	2,4%
	Total	109	4,9%	56	53	51%	49%	2,5%	2,4%
Mulheres	1ª Classe	144	6,5%	140	4	97%	3%	6,3%	0,2%
	2ª Classe	93	4,2%	80	13	86%	14%	3,6%	0,6%
	3ª Classe	165	7,4%	76	89	46%	54%	3,4%	4,0%
	Tripulação	23	1,0%	20	3	87%	13%	0,9%	0,1%
	Total	425	19,1%	316	109	74%	26%	14,2%	4,9%
Homens	1ª Classe	175	7,9%	57	118	33%	67%	2,6%	5,3%
	2ª Classe	168	7,6%	14	154	8%	92%	0,6%	6,9%
	3ª Classe	462	20,8%	75	387	16%	84%	3,3%	17,4%
	Tripulação	885	39,8%	192	693	22%	78%	8,6%	31,2%
	Total	1 690	75,9%	338	1 352	20%	80%	15,2%	60,8%
Total		**2 224**	**100%**	**710**	**1 514**	**32%**	**68%**	**31,9%**	**68,1%**

1.3. Objetivo

O principal objetivo é explorar os dados e entender como eles se comportam, e chegar a conclusão de qual modelo mais se mostrou aderente ao caso, gerando análises sobre quais variáveis tiveram mais influencia na probabilidade dos passageiros sobreviverem, ou seja, quem teve mais chances de escapar com vida. Para isso elaboraremos um modelo que dê a previsão de sobrevivência para um passageiro. Vamos passar por todo o processo de criação de um modelo de aprendizagem de máquina no conjunto de dados do Titanic. A base fornece informações sobre o destino dos passageiros do navio, resumidas de acordo com a situação econômica (classe), sexo, idade e sobrevivência, raça/cor, se está sozinho ou não. No final contaremos com 3 modelos: Random Forest, Decision Tree e Logistic Regression.

2. Coleta de Dados

O processo de coleta de dados foi iniciado com arquivos da plataforma Kaggle “Titanic Machine Learning from Disaster” Além disso, foi utilizado para enriquecer a base de dados sobre os passageiros, de forma fictícia, da plataforma Fake Name Generator. A base do Kaggle contém os dados principais do nosso dataset, uma linha por pessoa com colunas: *number, sex, survived, nameset, title, name, age, pelas, sibsp, parch, ticket, fare, cabin, embarked*. A base dos passageiros contém as seguintes colunas: *number, streaddress, city, state, stageful, telephone number, tropical zodiac, ccnumber, full color, occupation, company, centimeters, kilograms.*

O campo *number* foi usado para fazer o join e juntar as 2 bases, ao longo do notebook, foram feitos diversos tratamentos, como a exclusão de campos que não seriam utilizados e o tratamento de campos com duplicidade e nulos. Ao juntar as duas bases temos um total de 27 colunas. O quadro 1 foi elaborado para que seja possível uma melhor visualização da estrutura de dados, contemplando o nome do campo, sua descrição e o tipo de dado nele contido.

Nome da coluna	Descrição	Tipo
Number	Número do passageiro	inteiro
Sex	Sexo do passageiro	string
Survived	Esta coluna diz se o passageiro sobreviveu ou não (1, 0)	inteiro
NameSet	A nacionalidade do passageiro	string
Title	Título da pessoa (Dr, Mr, Mrs, Ms)	string
Name	Nome do passageiro	string
Age	Idade do passageiro	float64
Pclass	Qual a classe o passageiro está (1 primeira classe, 2 segunda classe e 3 terceira classe)	inteiro
SibSp	Número de irmãos ou esposas no barco	inteiro
Parch	Número de pais ou filhos dentro do barco	inteiro
Ticket	Número do Ticket	string

Fare	Tarifa paga pelo passageiro	float64
Cabin	Número da cabine	string
Embarked	Porto onde o passageiro embarcou	string
Number	Número do passageiro	inteiro
StreetAddress	Endereço do passageiro	string
City	Cidade onde mora o passageiro	string
State	Unidade da Federação / Estado	string
StateFull	Estado onde mora o passageiro	string
TelephoneNumber	Telefone do passageiro	string
TropicalZodiac	Signo do passageiro	string
CCNumber	Número de identidade do passageiro	float64
FulColor	Cor / Raça do passageiro	string
Occupation	Profissão do passageiro	string
Company	Empresa a qual o passageiro trabalha	string
Centimeters	Altura do passageiro em centimetros	inteiro
Kilograms	Peso do passageiro	float64

Considerando que a maior parte das colunas são do tipo string, foi necessário, efetuar uma série de transformações dos dados para números, já que nosso algoritmo apenas entende valores numéricos, na sessão de tratamento de dados será mostrado todos esses tratamentos.

3. Processamento e Tratamento de Dados

Uma das primeiras etapas é garantir que os dados procedem de fontes seguras e confiáveis para o desenvolvimento do tratamento, exploração e análise de dados. Más qualidades de dados de referência tornam-se rapidamente uma grande fonte de erros e resultam em ineficiências nas análises.

Para o desenvolvimento desse projeto foi utilizado o Jupyter Notebook, o primeiro passo foi importar todas as bibliotecas necessárias para desenvolvimento dos modelos. Logo após, foram importadas as 2 bases utilizadas e realizamos um estudo para melhor entendimento de cada uma, como por exemplo, como os dados estão organizados e estruturados com a função head. Cada base tem no total 2000 linhas segundo a função *len*, nossa base do Kagge foi chamada como “df” e a base com dados fake para enriquecimento do dataset foi chamada de “dfo".

Importando dados

In [1]:
```
import pandas as pd
import numpy as np
import zipfile
import requests
from io import BytesIO
import os
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import KFold, cross_val_predict, cross_val_score
from sklearn.svm import SVC
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn import metrics
from sklearn.model_selection import ShuffleSplit

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

df = pd.read_csv('/Users/gabrielfelipepinheirocouto/Documents/arquivos_machine_learning_tcc/data/database_one.csv')
dfo = pd.read_csv('/Users/gabrielfelipepinheirocouto/Documents/arquivos_machine_learning_tcc/data/database_two.csv')
```

In [2]:
```
df.head(3)
```

Out[2]:

	Number	Sex	Survived	NameSet	Title	Name	Age	Pclass	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	male	0	American	Mr.	Gregory Akins	22.0	3	1	0	A/5 21171	7.2500	NaN	S
1	2	male	1	American	Mr.	Matthew Erazo	38.0	1	1	0	PC 17599	71.2833	C85	C
2	3	male	1	American	Mr.	Cornell Banks	26.0	3	0	0	STON/O2. 3101282	7.9250	NaN	S

In [3]:
```
dfo.head(3)
```

Out[3]:

	Number	StreetAddress	City	State	StateFull	TelephoneNumber	TropicalZodiac	CCNumber	FulColor	Occupation	Company	Centimeters	K
0	1	1554 Orchard Street	Eden Prairie	MN	Minnesota	952-828-5758	Sagittarius	4.485926e+15	black	Decommissioning and decontamination (D&D) worker	Monmax	188	
1	2	2826 Irving Road	Mansfield	OH	Ohio	740-485-4116	Gemini	5.121207e+15	black	Production, planning, and expediting clerk	Harold's	185	
2	3	3212 Emma Street	Miami	TX	Texas	806-868-2588	Capricorn	5.149671e+15	dark skinned	Coil finisher	Matrix Architectural Service	165	

In [6]:
```
len(df)
len(dfo)
```

Out[6]:
```
2000
```

Nessa etapa, foi desenvolvido um tratamento para corrigir os nomes das colunas e deixá-las todas com mesmo padrão, ao observar os nomes das colunas com a função *dfo.dtypes* percebeu-se algumas com letras maiúsculas e outras não e também com espaço desnecessário, uma função foi criada para padronizar todas as colunas e ajustar os nomes dos dois datasets. No comando seguinte é chamado a função *head* e nota-se a correção.

```
In [5]: df.dtypes
Out[5]: Number        int64
        Sex          object
        Survived      int64
        NameSet      object
        Title        object
        Name         object
        Age         float64
        Pclass        int64
        SibSp         int64
        Parch         int64
        Ticket       object
        Fare        float64
        Cabin        object
        Embarked     object
        dtype: object
```

```
In [6]: dfo.dtypes
Out[6]: Number               int64
        StreetAddress       object
        City                object
        State               object
        StateFull           object
        TelephoneNumber     object
        TropicalZodiac      object
        CCNumber           float64
        FulColor            object
        Occupation          object
        Company             object
        Centimeters          int64
        Kilograms          float64
        dtype: object
```

Desenvolvendo uma função, a fim de ajustar as colunas e torná-las padronizadas

```
In [9]: import re

        def correct_columns(col_name):
            return re.sub(r"[/| ]", "", col_name).lower()
```

Chamando função para ajuste nas colunas

```
In [10]: df.columns = [correct_columns(col) for col in df.columns]
```

```
In [11]: df.head(3)
```

Out[11]:

	number	sex	survived	nameset	title	name	age	pclass	sibsp	parch	ticket	fare	cabin	embarked
0	1	male	0	American	Mr.	Gregory Akins	22.0	3	1	0	A/5 21171	7.2500	NaN	S
1	2	male	1	American	Mr.	Matthew Erazo	38.0	1	1	0	PC 17599	71.2833	C85	C

```
In [12]: dfo.columns = [correct_columns(col) for col in dfo.columns]

In [13]: dfo.head(3)
```

Out[13]:

	number	streetaddress	city	state	statefull	telephonenumber	tropicalzodiac	ccnumber	fulcolor	occupation	company	centimeters	kilog
0	1	1554 Orchard Street	Eden Prairie	MN	Minnesota	952-828-5758	Sagittarius	4.485926e+15	black	Decommissioning and decontamination (D&D) worker	Monmax	188	
1	2	2826 Irving Road	Mansfield	OH	Ohio	740-485-4116	Gemini	5.121207e+15	black	Production, planning, and expediting clerk	Harold's	185	

No próximo passo vamos juntar os dois datasets em apenas um, para isso chamaremos a função *len* para medirmos o tamanho do dataset em linhas, é importante fazer essa validação pois iremos fazer um *join*, portanto ambos os dataset devem está na mesma granularidade de dados. Após isso chamamos a função *join* e juntamos os datasets com a formatação *inner*, quando especificamos o *join* pelo *inner* a chave seleciona todas as linhas de ambas as tabelas, desde que haja uma correspondência entre as colunas. Se houver registos na tabela "df" que não tenham correspondências em "dfo", esses pedidos não serão exibidos, os dois dois datasets se tornaram um, com o nome "df".

Validaçao e join entre os datasets

```
In [14]: len(dfo)
Out[14]: 2000

In [15]: len(df)
Out[15]: 2000

In [16]: union = df.join(dfo.set_index('number'), on='number', how='inner')
         df = union

In [17]: df.head(3)
```

Out[17]:

	number	sex	survived	nameset	title	name	age	pclass	sibsp	parch	...	state	statefull	telephonenumber	tropicalzodiac	ccnumber	fulcolor
0	1	male	0	American	Mr.	Gregory Akins	22.0	3	1	0	...	MN	Minnesota	952-828-5758	Sagittarius	4.485926e+15	black
1	2	male	1	American	Mr.	Matthew Erazo	38.0	1	1	0	...	OH	Ohio	740-485-4116	Gemini	5.121207e+15	black
2	3	male	1	American	Mr.	Cornell Banks	26.0	3	0	0	...	TX	Texas	806-868-2588	Capricorn	5.149671e+15	dark skinned

3 rows × 26 columns

Agora tratamos os dados NaN (nulos) para que estes não gerem interferência em nosso modelo. Com esse intuito chamamos a função *head.isnull* que verificará quantos dados NaN existem .Para termos uma visão melhor chamamos o *for,* dessa forma olharemos os dados NaN por colunas listadas.

```
In [79]: df.head(4).isnull
Out[79]: <bound method DataFrame.isnull of    number  sex  survived  age  pclass  sibsp  parch     fare  embarked  fulcolor  \
         0       0    0   22       3       1      0   7.2500         1         1
         1       0    1   38       1       1      0  71.2833         2         1
         2       0    1   26       3       0      0   7.9250         1         2
         3       0    1   35       1       1      0  53.1000         1         1

            familysize  Isalone  fareCat
         0           2        0        1
         1           2        0        4
         2           1        1        2
         3           2        0        4  >
```

```
In [21]: for field in df.columns:
             print(field, 'NaN:', df[field].isnull().sum())

number NaN: 0
sex NaN: 0
survived NaN: 0
nameset NaN: 0
title NaN: 0
name NaN: 0
age NaN: 399
pclass NaN: 0
sibsp NaN: 0
parch NaN: 0
ticket NaN: 0
fare NaN: 0
cabin NaN: 1538
embarked NaN: 4
streetaddress NaN: 0
city NaN: 0
state NaN: 0
statefull NaN: 1
telephonenumber NaN: 0
tropicalzodiac NaN: 0
ccnumber NaN: 0
fulcolor NaN: 0
occupation NaN: 0
company NaN: 2
centimeters NaN: 0
kilograms NaN: 0
```

Pela imagem acima, notamos que temos algumas colunas com NaN, iremos começar pela coluna “emberked”, substituindo as quatros linhas NaN por 0. A coluna “cabin” não será útil para nosso modelo, por essa razão será excluída usando for e a função drop.

```
In [18]: for dataset in [df]:
             dataset.loc[dataset['embarked'].isnull(), 'embarked'] = 0
```

Entendendo que os dados de cabine não são relavante para nosso modelo, portando iremos eliminar

```
In [19]: df = df.drop(['cabin'], axis=1)
```

Para o sexo masculino e sexo feminino iremos dar o valor de 1 e 0 já que nosso modelo, não suporta palavras, apenas números.

```
In [20]: df['sex'] = df['sex'].map( {'female': 1, 'male': 0} ).astype(int)
```

Nesse tratamento, precisamos corrigir os NaN do campo “age”. Para isso, primeiro vamos explorar os dados e colocar os filtros, com o objetivo de entender esses dados que serão tratados. Para encontrar o que precisamos fizemos o uso da função loc. Após entendermos os dados, criamos o *for* com variáveis para conseguirmos a média de idade de pessoas com as mesmas características e vincula a pessoas com campo “age” NaN. Após o *for,* chamamos a função *loc, isnull* e *len* a fim de validar se o tratamento trouxe uma solução para o erro.

```
In [74]:
guess_ages = np.zeros((2,3))

for dataset in [df]:
    for sex in range(0, 2):
        for pclass in range(0, 3):
            guess_df = dataset[(dataset['sex'] == sex) & (dataset['pclass'] == pclass + 1)]['age'].dropna()

            age_guess = guess_df.median()
            print('sex', sex, 'pclass', pclass)
            print(len(guess_df), age_guess)

            # Converte para a idade mais proxima (para nao ficar 0.4 ou 0.333)
            guess_ages[sex, pclass] = int( age_guess / 0.5 + 0.5 ) * 0.5

    for sex in range(0, 2):
        for pclass in range(0, 3):
            dataset.loc[ (dataset.age.isnull()) & (dataset.sex == sex) & (dataset.pclass == pclass + 1), 'age'] = guess

    dataset['age'] = dataset['age'].astype(int)
```

```
In [23]: df.loc[df['age'].isnull()]
```

Out[23]:

number	sex	survived	nameset	title	name	age	pclass	sibsp	parch	...	state	statefull	telephonenumber	tropicalzodiac	ccnumber	fulcolor	occupation

0 rows × 25 columns

```
In [24]: print('df:', len(df.loc[df['age'].isnull()]))
df: 0

In [25]: for field in df.columns:
             print(field, 'NaN:', df[field].isnull().sum())
```

Ao olhar para nossa base, ainda temos as duas linhas NaN na coluna “company”, usamos a função *drop* para excluir as duas linhas, também a função *len* para validação do número de linhas do nosso dataset.

```
In [25]: for field in df.columns:
             print(field, 'NaN:', df[field].isnull().sum())

number NaN: 0
sex NaN: 0
survived NaN: 0
nameset NaN: 0
title NaN: 0
name NaN: 0
age NaN: 0
pclass NaN: 0
sibsp NaN: 0
parch NaN: 0
ticket NaN: 0
fare NaN: 0
embarked NaN: 0
streetaddress NaN: 0
city NaN: 0
state NaN: 0
statefull NaN: 1
telephonenumber NaN: 0
tropicalzodiac NaN: 0
ccnumber NaN: 0
fulcolor NaN: 0
occupation NaN: 0
company NaN: 2
centimeters NaN: 0
kilograms NaN: 0
```

Notamos que as colunas statefull e company tem linhas NaN, como se trata de uma linha, iremos exclui-las da base

```
In [26]: len(df)
Out[26]: 2000

In [27]: df = df.dropna()

In [28]: len(df)
Out[28]: 1997
```

Iremos iniciar nossa transformação de dados e criação de novas features para desenvolvimento do nosso modelo. A primeira transformação que iremos fazer é na coluna de “title”, transformaremos cada título em um número, indo de 1 até 4, usamos o *for* e a função *crosstab* Esta função é usada para obter uma “sensação” (visão) inicial dos dados. Podemos validar algumas hipóteses básicas. Na imagem abaixo, mostra bem qual o formato dos dados apresentados, usando a função *crosstab.*

```
In [31]: pd.crosstab(df["title"], df['sex'])
```

Out[31]:

sex	0	1
title		
Dr.	33	29
Mr.	975	0
Mrs.	0	454
Ms.	0	506

Transformando a coluna sex em números

```
In [32]: title_mapping = {"Dr.": 1, "Mr.": 2, "Mrs.": 3, "Ms.": 4}

for dataset in [df]:
    dataset['title'] = dataset['title'].map(title_mapping)
    dataset['title'] = dataset['title'].fillna(0)
```

```
In [33]: pd.crosstab(df['title'], df['sex'])
```

Out[33]:

sex	0	1
title		
1	33	29
2	975	0
3	0	454
4	0	506

```
In [34]: df.head(3)
```

Out[34]:

	number	sex	survived	nameset	title	name	age	pclass	sibsp	parch	...	state	statefull	telephonenumber	tropicalzodiac	ccnumber	fulcolor	De
0	1	0	0	American	2	Gregory Akins	22	3	1	0	...	MN	Minnesota	952-828-5758	Sagittarius	4.485926e+15	black	d
1	2	0	1	American	2	Matthew Erazo	38	1	1	0	...	OH	Ohio	740-485-4116	Gemini	5.121207e+15	black	
2	3	0	1	American	2	Cornell Banks	26	3	0	0	...	TX	Texas	806-868-2588	Capricorn	5.149671e+15	dark skinned	

3 rows × 25 columns

O passageiro está sozinho ou não? Esta é uma característica importante, pois algumas pessoas deixaram de subir nos botes salva-vidas para ir buscar algum parente dentro do navio e acabaram não sobrevivendo. Faremos o tratamento seguindo as seguintes etapas: Vamos pegar o tamanho da família a qual o passageiro pertence, depois criaremos uma nova feature chamada isalone indicando se ele está sozinho e por último rodamos os casos em que o tamanho da família for igual a 1, aqui consideraremos que ele está acompanhado. Usamos função *loc* e soma na variável.

```
In [35]: df['familysize'] = df['sibsp'] + df['parch'] + 1

         df['Isalone'] = 0

         df.loc[df['familysize'] == 1, 'Isalone'] = 1
```

```
In [36]: df.head(3)
```

Out[36]:

age	pclass	sibsp	parch	...	telephonenumber	tropicalzodiac	ccnumber	fulcolor	occupation	company	centimeters	kilograms	familysize	Isalone
22	3	1	0	...	952-828-5758	Sagittarius	4.485926e+15	black	Decommissioning and decontamination (D&D) worker	Monmax	188	68.3	2	0
38	1	1	0	...	740-485-4116	Gemini	5.121207e+15	black	Production, planning, and expediting clerk	Harold's	185	84.3	2	0
26	3	0	0	...	806-868-2588	Capricorn	5.149671e+15	dark skinned	Coil finisher	Matrix Architectural Service	165	71.1	1	1

Continuando com o tratamento, iremos explorar e excluir algumas colunas que não serão relevantes para nosso modelo, sendo elas: “nameset”, “telephonenumber”, “tropicalzodiac”, usamos a função *drop* para essa finalidade.

```
In [37]: np.unique(df["nameset"])
Out[37]: array(['American'], dtype=object)

In [38]: df = df.drop(['nameset'], axis=1)

In [39]: np.unique(df["telephonenumber"])
Out[39]: array(['201-288-1555', '201-319-7369', '201-370-0851', ...,
                '989-769-4784', '989-894-3207', '989-942-5911'], dtype=object)

In [40]: df = df.drop(['telephonenumber'], axis=1)

In [41]: np.unique(df["tropicalzodiac"])
Out[41]: array(['Aquarius', 'Aries', 'Cancer', 'Capricorn', 'Gemini', 'Leo',
                'Libra', 'Pisces', 'Sagittarius', 'Scorpio', 'Taurus', 'Virgo'],
               dtype=object)

In [42]: df = df.drop(['tropicalzodiac'], axis=1)

In [43]: df.head(3)
```

Out[43]:

	number	sex	survived	title	name	age	pclass	sibsp	parch	ticket	...	state	statefull	ccnumber	fulcolor	occupation	company	cent
0	1	0	0	2	Gregory Akins	22	3	1	0	A/5 21171	...	MN	Minnesota	4.485926e+15	black	Decommissioning and decontamination (D&D) worker	Monmax	
1	2	0	1	2	Matthew Erazo	38	1	1	0	PC 17599	...	OH	Ohio	5.121207e+15	black	Production, planning, and expediting clerk	Harold's	
2	3	0	1	2	Cornell Banks	26	3	0	0	STON/O2. 3101282	...	TX	Texas	5.149671e+15	dark skinned	Coil finisher	Matrix Architectural Service	

3 rows × 24 columns

Na coluna “fulcolor” iremos atribuir um número para cada string diferente, assim deixaremos a coluna aderente ao modelo. Começaremos usamos a função *unique* para trazer os valores únicos, e depois usando a variável em biblioteca junto com o *for* para efetuar a transformação.

```
In [44]: np.unique(df["fulcolor"])
Out[44]: array(['black', 'dark skinned', 'white'], dtype=object)

In [45]: title_mapping = {"black": 1, "dark skinned": 2, "white": 3}

         for dataset in [df]:
             dataset['fulcolor'] = dataset['fulcolor'].map(title_mapping)
             dataset['fulcolor'] = dataset['fulcolor'].fillna("")

In [46]: np.unique(df["fulcolor"])
Out[46]: array([1, 2, 3])
```

Em nosso dataset ainda existem algumas colunas que não terão utilidade no modelo, nesse caso iremos excluí-las deixando apenas dados úteis para o desenvolvimento; as colunas excluídas serão: “ticket”, “state”, “statefull”, “ccnumber”, “occupation”, “company”, “city”, “streetaddress”, “name”, “centimeters”, “kilograms”, “number”, “title”. Usamos a função *drop* para excluir, é notável que o dataset foi enxugado após a exclusão, ficando praticamente com colunas numéricas. A função *info*, traz as colunas e o type de cada uma.

```
In [48]: df = df.drop(['ticket',"state","statefull","ccnumber","occupation","company","city","streetaddress","name", "centimeter
```

```
In [49]: df.head(3)
```

Out[49]:

	sex	survived	age	pclass	sibsp	parch	fare	embarked	fulcolor	familysize	Isalone
0	0	0	22	3	1	0	7.2500	S	1	2	0
1	0	1	38	1	1	0	71.2833	C	1	2	0
2	0	1	26	3	0	0	7.9250	S	2	1	1

```
In [50]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1997 entries, 0 to 1999
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sex         1997 non-null   int64
 1   survived    1997 non-null   int64
 2   age         1997 non-null   int64
 3   pclass      1997 non-null   int64
 4   sibsp       1997 non-null   int64
 5   parch       1997 non-null   int64
 6   fare        1997 non-null   float64
 7   embarked    1997 non-null   object
 8   fulcolor    1997 non-null   int64
 9   familysize  1997 non-null   int64
 10  Isalone     1997 non-null   int64
dtypes: float64(1), int64(9), object(1)
memory usage: 187.2+ KB
```

Os algoritmos de Machine Learning podem se dar melhor com campos que possuem poucos valores do que com campos que possuem milhares de informações diferentes, então categorizar as taxas pagas colocando este valor em uma categoria pode ser interessante e existem algumas formas de fazer isso. Iremos primeiro usar a função *describe* para ver os dados numéricos especificamente da coluna, depois o *for* para criação da nova features.

```
In [51]: df['fare'].describe()
```

```
Out[51]: count    1997.000000
         mean       31.504249
         std        48.000765
         min         0.000000
         25%         7.895800
         50%        14.454200
         75%        30.695800
         max       512.329200
         Name: fare, dtype: float64
```

Ja excluimos as colunas que não devem fazer parte do nosso modelo, agora vamos continuar a trasformação de dados

```
In [52]: for dataset in [df]:
             dataset['fareCat'] = 0
             dataset.loc[dataset.fare < 7.91, 'fareCat'] = 1
             dataset.loc[(dataset.fare >= 7.91) & (dataset.fare <= 14.45), 'fareCat'] = 2
             dataset.loc[(dataset.fare >= 14.45) & (dataset.fare <= 31), 'fareCat'] = 3
             dataset.loc[dataset.fare > 31, 'fareCat'] = 4

         df.head(3)
```

Out[52]:

	sex	survived	age	pclass	sibsp	parch	fare	embarked	fulcolor	familysize	lsalone	fareCat
0	0	0	22	3	1	0	7.2500	S	1	2	0	1
1	0	1	38	1	1	0	71.2833	C	1	2	0	4
2	0	1	26	3	0	0	7.9250	S	2	1	1	2

Por último ainda temos a coluna “embarked” com valores de string, iremos aplicar o mesmo tratamento usado anteriormente em outras colunas, ou seja, usaremos a função *unique* para entender os valores, criaremos uma variável de biblioteca e depois usaremos o for para a transformação. Dessa forma, finalizamos nosso tratamento de dados e criação de features para nosso modelo.

```
In [53]: df.embarked.unique()
```

```
Out[53]: array(['S', 'C', 'Q', 0], dtype=object)
```

```
In [54]: embarked_mapping = {"S": 1, "C": 2, "Q": 3, 0: 0}
         for dataset in [df]:
             dataset['embarked'] = dataset['embarked'].map(embarked_mapping)

         df.head(3)
```

Out[54]:

	sex	survived	age	pclass	sibsp	parch	fare	embarked	fulcolor	familysize	lsalone	fareCat
0	0	0	22	3	1	0	7.2500	1	1	2	0	1
1	0	1	38	1	1	0	71.2833	2	1	2	0	4
2	0	1	26	3	0	0	7.9250	1	2	1	1	2

4. Análise e Exploração dos Dados

Nessa etapa, iremos explorar os dados com o objetivo de identificar padrões, relações e tendências. A finalidade aqui é tirar insights através da análise sobre o dataset. Primeiro passo usaremos as bibliotecas matplotlib e seaborn para geração dos nossos gráficos.

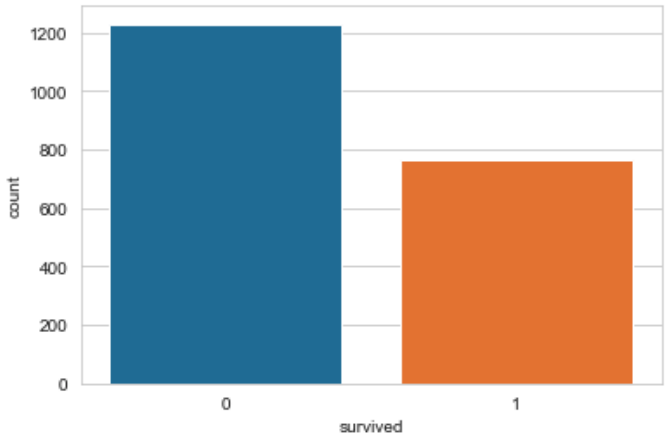
Primeira visão que teremos é o percentual de sobreviventes. Vamos descobrir a distribuição dos que sobreviveram (1) e os que morreram (0), para isso usaremos a função *len* e a função da *countplot. A*o observarmos os dados, notamos o percentual de 38% de sobrevivência.

Percentual de sobreviventes

```
In [56]: print('Survival percentage:', len(df[df.survived == 1]) / len(df))
         sns.countplot(x="survived", data=df)

         Survival percentage: 0.38407611417125687

Out[56]: <AxesSubplot:xlabel='survived', ylabel='count'>
```



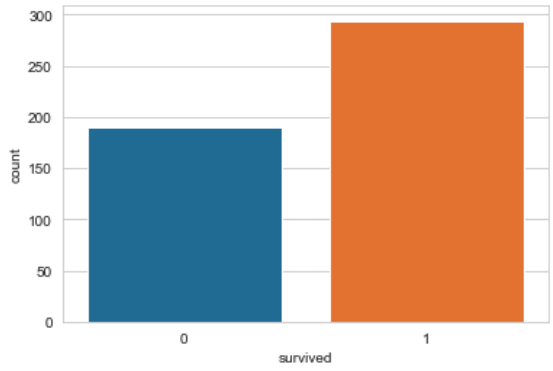
Qual a porcentagem de sobrevivência das pessoas que estavam na primeira classe? Ao olharmos o gráfico conseguimos dimensionar a quantidade de sobreviventes e o percentual de 14%.

Porcentagem de sobrevivência de pessoas que estavam na primeira classe

```
In [77]: print('Survival percentage:', len(df[(df.pclass == 1) & (df.survived == 1 )]) / len(df))
         sns.countplot(x="survived", data=df[df.pclass == 1])

         Survival percentage: 0.1472208312468703

Out[77]: <AxesSubplot:xlabel='survived', ylabel='count'>
```



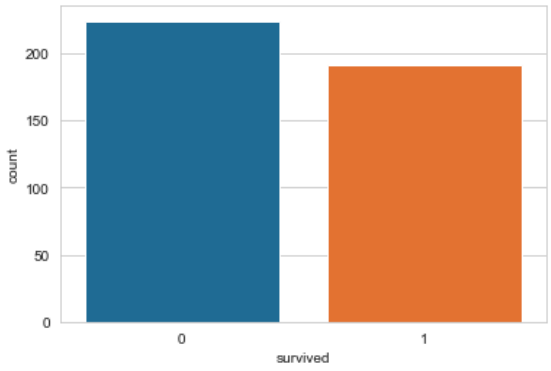
Qual a porcentagem de sobrevivência das pessoas que estavam na segunda classe? O número baixa para 09%

Porcentagem de sobrevivência de pessoas que estavam na segunda classe

```
In [78]: print('Survival percentage:', len(df[(df.pclass == 2) & (df.survived == 1 )]) / len(df))
         sns.countplot(x="survived", data=df[df.pclass == 2])

         Survival percentage: 0.09564346519779669

Out[78]: <AxesSubplot:xlabel='survived', ylabel='count'>
```



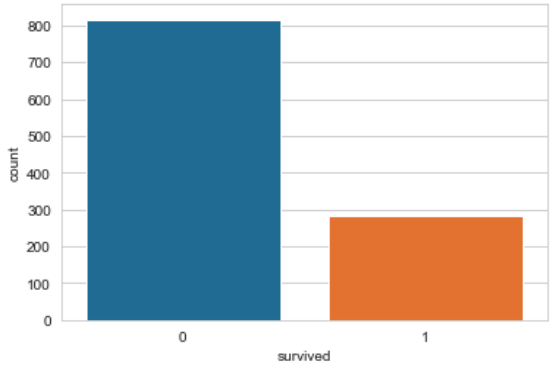
Qual a porcentagem de sobrevivência das pessoas que estavam na terceira classe? O número sobe para 14%

Porcentagem de sobrevivência de pessoas que estavam na terceira classe

```
In [79]: print('Survival percentage:', len(df[(df.pclass == 3) & (df.survived == 1 )]) / len(df))
         sns.countplot(x="survived", data=df[df.pclass == 3])

         Survival percentage: 0.1412118177265899

Out[79]: <AxesSubplot:xlabel='survived', ylabel='count'>
```



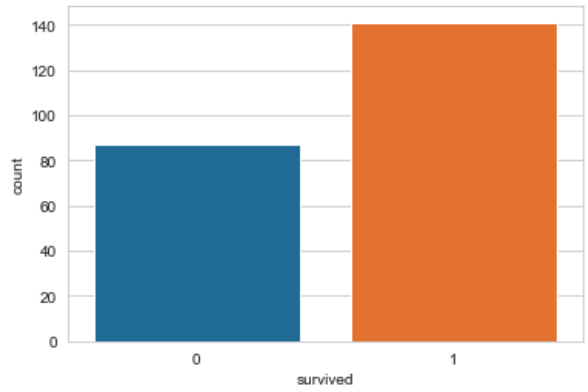
Olhando agora especificamente para mulheres que estiveram na primeira classe, temos 07% de percentual de sobrevivência.

Entendendo se mulheres de primeira classe sobreviveram

```
In [83]: print(len(df[(df.pclass == 1) & (df.sex == 1 ) & (df.survived == 1)]) / len(df))
         sns.countplot(x="survived", data=df[(df.pclass == 1) & (df.sex == 1 )])

         0.07060590886329494

Out[83]: <AxesSubplot:xlabel='survived', ylabel='count'>
```



Exploramos os dados com filtros para trazer apenas mulheres, que não sobreviveram e que estiveram na primeira classe, segundo nosso gráfico acima.

Entendendo por que algumas das mulheres de primeira classe morreram

```
In [61]: df[(df.sex == 1) & (df.survived == 0) & (df.pclass == 1)]
```

Out[61]:

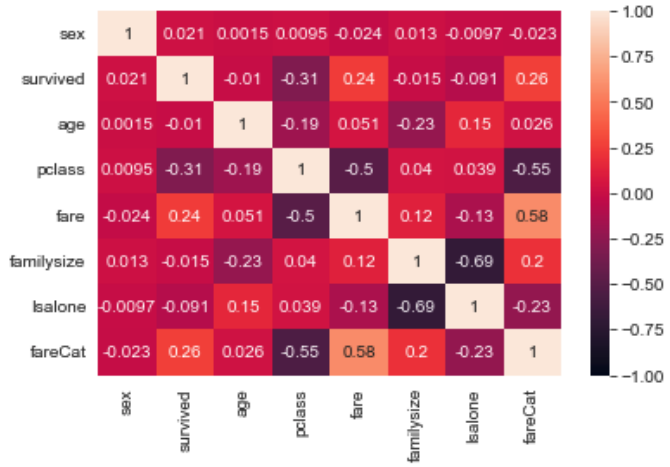
	sex	survived	age	pclass	sibsp	parch	fare	embarked	fulcolor	familysize	lsalone	fareCat
34	1	0	28	1	1	0	82.1708	2	2	2	0	4
35	1	0	42	1	1	0	52.0000	1	3	2	0	4
64	1	0	34	1	0	0	27.7208	2	2	1	1	3
102	1	0	21	1	0	1	77.2875	1	3	2	0	4
124	1	0	54	1	0	1	77.2875	1	3	2	0	4
...	...	...	...	...	...	...	...	...	...	...	...	...
1942	1	0	25	1	1	1	7.8958	1	3	3	0	1
1943	1	0	41	1	0	0	10.5167	3	1	1	1	2
1969	1	0	20	1	0	0	13.0000	1	1	1	1	2
1983	1	0	34	1	0	0	7.8958	1	3	1	1	1
1988	1	0	20	1	0	1	10.5000	1	3	2	0	2

87 rows × 12 columns

Uma outra forma de analisarmos esses dados, é com matriz de correlação, excluímos as colunas: “sibsp”, “parch”, “embarked”, “fulcolor”. O motivo de excluímos essas colunas, é para que nossa matriz não esteja poluída ao ponto de não enxergarmos os números e correções.

```
In [98]: pt = df.drop(['sibsp',"parch","embarked","fulcolor"], axis=1)
```

```
In [99]: sns.heatmap(pt.corr(), annot=True, vmin=-1, vmax=1);
```



Para essa exploração de dados, usamos as bibliotecas matplotlib e seaborn. O matplotlib é uma biblioteca com recursos para a geração de gráficos 2D a partir de arrays. Gráficos comuns podem ser criados com alta qualidade a partir de comandos simples, inspirados nos comandos gráficos do Python. O conjunto de funções disponível em matplotlib.pyplot permitem que você crie uma figura, uma área para exibir o gráfico na figura, desenhe linhas na área do gráfico, decore o gráfico com rótulos, etc. A sintaxe utilizada é semelhante ao matlab. Usamos o gráfico tradicional de barras, para identificação da quantidade de sobreviventes e mortos. Ao observarmos o primeiro gráfico e o percentual setado, notamos que 38% das pessoasa bordo sobreviveu a tragédia, dissemos o nível aao olhamos esses mesmo dados, mas agora olhando apenas para pessoas da primeira classe representa 14%, ou seja, 36% do total de sobreviventes estava nessa classe, esse fator é influenciado porque pessoas nas maiores classes tinha prioridade nos botes de salvas vidas.

9% é o número de sobreviventes na segunda classe, onde as pessoas também tinham prioridade nos botes, ao olharmos o percentual de 3 classes, temos 14% de sobrevivência. Exploramos mais os sobreviventes da primeira classe, agora filtrando para olharmos apenas pessoas do gênero feminino, que é representado por 7%. Ou seja, 50% dos sobreviventes da primeira classe são do gênero feminino. Crianças e mulheres tinham prioridades nos botes.

Exploramos também o gráfico de matriz de correlação, usamos a biblioteca soaborn, seaborn é uma biblioteca de visualização de dados Python baseada em matplotlib. Ele fornece uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos. Ao explorarmos o gráfico, excluímos as colunas: “sibsp”, “parch”, “embarked”, "fullcolor" no objetivo de deixar nossa visualização mais limpa, deixaremos os dados que entendemos ter mais relação. O comando usado *pt.corr* utiliza por padrão a correlação de Person.

O coeficiente de correlação de Pearson não tem esse nome por acaso. É comum atribuir exclusivamente a Karl Pearson o desenvolvimento dessa estatística, no entanto, como bem lembrou Stanton (2001), a origem desse coeficiente remonta o trabalho conjunto de Karl Pearson e Francis Galton (Stanton, 2001: 01). Garson (2009) afirma que correlação “é uma medida de associação bivariada (força) do grau de relacionamento entre duas variáveis”. Para Moore (2007), “A correlação mensura a direção e o grau da relação linear entre duas variáveis quantitativas” (Moore, 2007: 100/101). Em uma frase: o coeficiente de correlação de Pearson (r) é uma medida de associação linear entre variáveis. Sua fórmula é a seguinte:

$$r = \frac{1}{n-1}\sum\left(\frac{xi - \bar{X}}{sx}\right)\left(\frac{yi - \bar{Y}}{sy}\right)$$

O coeficiente de correlação Pearson (r) varia de -1 a 1. O sinal indica direção positiva ou negativa do relacionamento e o valor sugere a força da relação entre as variáveis. Uma correlação perfeita (-1 ou 1) indica que o escore de uma variável pode ser determinado exatamente ao se saber o escore da outra. No outro oposto, uma correlação de valor zero indica que não há relação linear entre as variáveis.

Ao olharmos a correlação do gráfico, notamos os valores de “fare” fortemente relacionado “farecat” isso se dá porque criamos uma feature, com range do valor pago da passagem. Notamos também uma correlação entre “survided” com o range de “farecat”, pois quanto maior é o valor pago, melhor é a classe no navio, por consequência pessoas nas melhores classes além de mulheres e crianças, tinham prioridade nos botes salva-vidas. Quando olhamos para os dados com baixa correlação (números negativos), “isalone” tem baixa correlação com “survided” ou seja, a pessoa estar sozinha não impacta a possibilidade de sobrevivência, pois segundo os dados com boa correlação, a possibilidade de sobrevivência está relacionada ao perfil de pessoa.

5. Criação de Modelos de Machine Learning

Para desenvolvimento dos modelos escolhemos três algoritmos: Random Forest, Decision Tree, Logistic Regression. São algoritmos já conhecidos dentro da comunidade de Machine Learning e apresentados na disciplina do professor Hugo de Paula ao longo do curso.

Para começarmos, separaremos a base de teste e a base de treino, 70% para base de treino e 30% para base de teste. Após separarmos as bases, iremos dividir em 2 bases novamente, uma com os dados dos sobreviventes somente, e outra sem essa informação, a intenção é treinarmos os modelos e testarmos a aderência de cada algoritmo. Usamos a função *random.rand* da biblioteca numpy para separarmos as bases, também criamos nossas variáveis com o nome de cada modelo.

Desenvolvimento de modelos de Machine Learning

Explorando os dados já tratados e prontos para serem usados e separando nossa base de teste e treinamento

In [66]:
```
msk = np.random.rand(len(df)) < 0.7
train = df[msk]
test = df[~msk]
```

In [67]:
```
X_train = train.drop(["survived"], axis=1)
Y_train = train["survived"]
X_test  = test.drop("survived", axis=1)
Y_test  = test["survived"]
```

Modelos de Machine Learning

In [85]:
```
modelos = [LogisticRegression(solver='liblinear'), RandomForestClassifier(n_estimators=400, random_state=42),
           DecisionTreeClassifier(random_state=42)]
```

Validação cruzada

In [76]:
```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

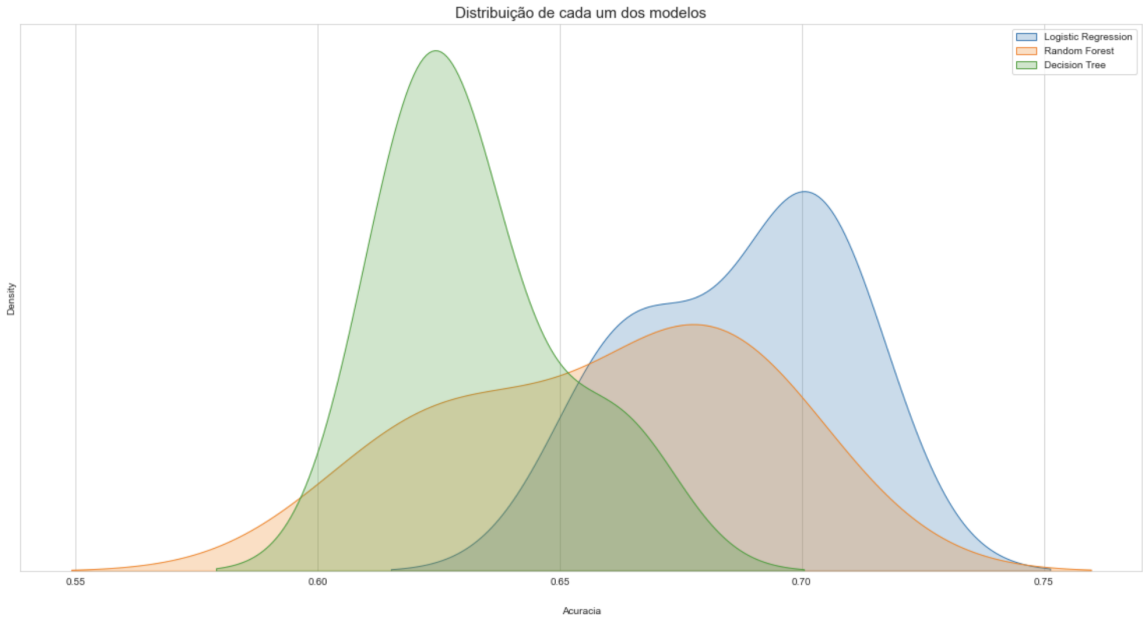
In [77]:
```
mean=[]
std=[]
for model in modelos:
  result = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy', n_jobs=-1)
  mean.append(result)
  std.append(result)
```

In [79]:
```
classificadores = ['Logistic Regression', 'Random Forest', 'Decision Tree']

plt.figure(figsize=(20, 10))
for i in range(len(mean)):
  sns.distplot(mean[i], hist=False, kde_kws={"shade":True})

plt.title("Distribuição de cada um dos modelos", fontsize=15)
plt.legend(classificadores)
plt.xlabel("Acuracia", labelpad=20)
plt.yticks([])
```

Utilizamos o Cross-validation para validação de dados e análise de aderência sobre os modelos escolhidos. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. Esta técnica é amplamente empregada onde o objetivo da modelagem é a predição. Busca-se então estimar o quão preciso é este modelo na prática, ou seja, o seu desempenho para um novo conjunto de dados. Uma das maneiras de fazer a divisão desses dados é usando o método holdout, ele consiste em dividir os dados em 70-30 de maneira aleatória, ou seja, 70% dos dados para treino e 30% para teste. Usamos a função *cross_val_score* e *kfold* da biblioteca sklearn para nosso cross-validation e também a função *plt* da biblioteca matplotlib para setar nosso gráfico e analisar a média de cada modelo, gerando o score final.



Ao analisar o gráfico nota-se que o modelo Random Forest tem o melhor score com valor acima de 0.75, Logistic Regression tem o segundo melhor score com 0.75 e Decison Tree tem o pior valor com 0.70. Ou seja, dos 3 modelos testados, Random Forest apresenta o melhor valor através do cross-validation.

Após essa etapa cada modelo recebe nomes de variáveis e na sequência fez-se o uso do método *fit* para treinar os modelos, pedaço X e Y da base *train*. Foi utilizado o método *predict_proba* para retornar estimativas de probabilidade na base *test* pedaço X, após esse processamento a função *argmax* foi acionada da biblioteca numpy para trazer o índice da matriz com a maior probabilidade prevista, retornando entre 1 e 0 para sobrevivência ou morte.

```
In [72]: rf_clf = RandomForestClassifier(n_estimators=400, n_jobs=-1, random_state=42)
         dt_clf = DecisionTreeClassifier(random_state=42)
         lr_clf = LogisticRegression(solver='liblinear')

         rf_clf.fit(X_train, Y_train)
         dt_clf.fit(X_train, Y_train)
         lr_clf.fit(X_train, Y_train)

Out[72]: LogisticRegression(solver='liblinear')

In [73]: rf_prob = rf_clf.predict_proba(X_test)
         dt_prob = dt_clf.predict_proba(X_test)
         lr_prob = lr_clf.predict_proba(X_test)

         rf_preds = np.argmax(rf_prob, axis=1)
         dt_preds = np.argmax(dt_prob, axis=1)
         lr_preds = np.argmax(lr_prob, axis=1)
```

Para a visualização dos resultados dos modelos fez-se o uso da matriz de confusão, responsável por calcular a precisão e detalhar os resultados.

No campo do aprendizado de máquina e especificamente no problema de classificação estatística, uma matriz de confusão, também conhecida como matriz de erro, é um layout de tabela específico que permite a visualização do desempenho de um algoritmo, tipicamente de aprendizado supervisionado (em aprendizado não supervisionado, geralmente é chamado de matriz de correspondência). Cada linha da matriz representa as instâncias de uma classe real enquanto cada coluna representa as instâncias de uma classe predita, ou vice-versa – ambas as variantes são encontradas na literatura. O nome deriva do fato de que torna fácil ver se o sistema está confundindo duas classes (ou seja, comumente rotulado, erroneamente, uma com a outra).

É um tipo especial de tabela de contingência, com duas dimensões ("real" e "prevista") e conjuntos idênticos de "classes" em ambas as dimensões (cada combinação de dimensão e classe é uma variável na tabela de contingência).

Usou-se as bases *test* e a variável já com o cálculo da probabilidade, pedaço X e Y. Depois dividiu-se pelo valor da soma total para se obter os resultados em percentuais. Para design e criação de layout com paletas de cores foi utilizada da biblioteca matplotlib a função *subplots* e também a função *heatmap* para o mapa de calor categórico.

```
In [77]: cm1 = metrics.confusion_matrix(Y_test, rf_preds)
         cm1 = cm1.astype('float') / cm1.sum(axis=1)[:, np.newaxis]

         cm2 = metrics.confusion_matrix(Y_test, dt_preds)
         cm2 = cm2.astype('float') / cm2.sum(axis=1)[:, np.newaxis]

         cm3 = metrics.confusion_matrix(Y_test, lr_preds)
         cm3 = cm3.astype('float') / cm3.sum(axis=1)[:, np.newaxis]
```

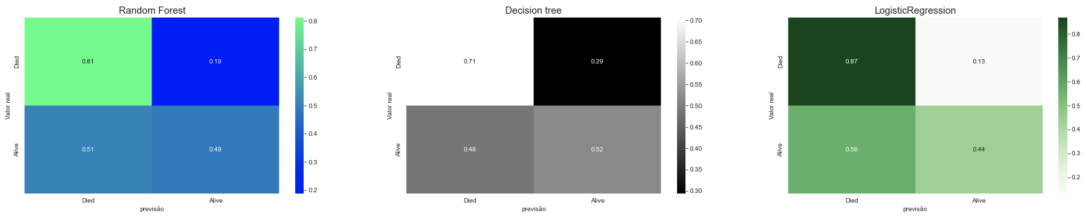
```
In [79]: classes=["Died", "Alive"]
         f, ax = plt.subplots(1, 3,  figsize=(30, 5))

         ax[0].set_title("Random Forest", fontsize=15.)
         sns.heatmap(pd.DataFrame(cm1, index=classes, columns=classes),
                     cmap='winter', annot=True, fmt='.2f', ax=ax[0]).set(xlabel="previsão", ylabel="Valor real")

         ax[1].set_title("Decision tree", fontsize=15.)
         sns.heatmap(pd.DataFrame(cm2, index=classes, columns=classes),
                     cmap='gray', annot=True, fmt='.2f', ax=ax[1]).set(xlabel="previsão", ylabel="Valor real")

         ax[2].set_title("LogisticRegression", fontsize=15.)
         sns.heatmap(pd.DataFrame(cm3, index=classes, columns=classes),
                     cmap='Greens', annot=True, fmt='.2f', ax=ax[2]).set(xlabel="previsão", ylabel="Valor real")
```

```
Out[79]: [Text(0.5, 25.0, 'previsão'), Text(1431.6470588235295, 0.5, 'Valor real')]
```



Dessa forma concluímos todo o processamento com a apresentação final dos resultados de cada modelo: Random Forest, Decision Tree, Logistic Regression. cada modelo teve sua particularidade e resultado único na análise que será apresentada no próximo capítulo.

6. Apresentação dos Resultados

O naufrágio do RMS Titanic é um dos mais famosos naufrágios da história. Em 15 de abril de 1912, durante sua viagem inaugural, o Titanic afundou depois de colidir com um iceberg, matando 1502 de 2224 passageiros e tripulantes. Essa tragédia chocou a comunidade internacional e levou a elaboração de melhores normas de segurança para os navios.

Uma das razões pelas quais o naufrágio causou tamanha perda de vidas foi que não havia botes salva-vidas suficientes para os passageiros e a tripulação. Embora houvesse algum elemento de sorte envolvido na sobrevivência do naufrágio, alguns grupos de pessoas tinham maior probabilidade de sobreviver do que outros, como mulheres, crianças e a classe alta.

Primeiro exploramos os dados, nesta análise exploratória verificamos padrões nos dados levantando estatísticas descritivas, tabelas cruzadas, dados faltantes e dados anômalos. De acordo com a análise exploratória e o entendimento do problema, a primeira hipótese é que os sobreviventes eram, em sua maioria, mulheres e crianças de classe alta e praticamente com relações familiares de primeiro grau; a segunda hipótese é que pessoas da primeira classe tiveram uma chance maior de sobreviver, dado ao fácil acesso aos botes salva-vidas. Foi identificado também que a cor da pele ou características físicas não impactavam na probabilidade de sobrevivência.

Foi feito o uso, em uma parte da análise exploratória, da correlação de Pearson. Ela avalia a relação linear entre duas variáveis contínuas. Uma relação é linear quando a mudança em uma variável é associada a uma mudança proporcional na outra variável. Nessa análise notamos o campo *fare* com uma boa correlação com o campo de sobrevivência, o campo representa o valor pago pela passagem, logo quanto mais o valor maior a classe, maior a classe, maior a probabilidade de sobrevivência. No campo age, foi identificado uma baixa correlação, pois a preferência era por crianças a mulheres, ou seja, mesmo tendo homens com idades mais novas, eles não eram prioridades no momento de evacuação do navio.

Usamos o Data Science Workflow Canvas para auxiliar no fluxo do projeto, respondendo às seguintes perguntas: Qual é o Problema? Quais os resultados? Quais fontes de dados usadas? Quais modelos foram escolhidos? Quais métricas foram avaliadas? Qual foi a preparação dos dados? Abaixo a tabela:

Title: **Análise Preditiva**		
Problem Statement What problem are you trying to solve? What larger ssues do the problem address? **Prever se o passageiro do navio pode morrer ou não ?** **Compreender o perfil de pessoas com maior probablilidade de morrer ?** **Entender o motivo de morte e sobrevivencia dos passageiros ?**	**Outcomes/Predictions** What prediction(s) are you trying to make? Identify applicable predictor (X) or target (y) **Identificar variáveis que impactam na probabilidade.** **Correlação entre variáveis.** **Percentual de sobrevivência.**	**Data Acquisition** Where are you sourcing your data from? Is there enough data? Can you work with it? **Dados extradios da plataforma Kaggle e enriquecidos com dados da plataforma FakeName Generator.**
Modeling What models are appropriate to use given your outcomes? **Supervid Machine Learning** **Logistic Regression foi o modelo com o melhor resultado.**	**Model Evaluation** How can you evaluate your model's performance? **Cross Validation** **Confusion Matrix**	**Data Preparation** What do you need to do to your data in order to run your model and achieve your outcomes? **Modelar os dados e excluir partes desnecessárias.** **Juntar os dois datasets em apenas um, e separar a parte de treino e outra de teste.** **Tratar dados com erros e criar novas features.**

Observando a análise feita através da matriz de confusão, notamos os resultados dos três modelos, cada um com seu valor e aderência. As cores foram escolhidas aleatoriamente, assim como os modelos.

Estes modelos apresentaram valores bem próximos, porém o que apresentou pior resultado foi o Logistic Regression, apesar de no primeiro eixo ter a assertividade na previsão com 84% de pessoas que morreram, apresentou o valor de 40% de pessoas que sobreviveram, valor abaixo de 50%. Outro modelo que não apresentou valores tão bons, foi o Random Forest com 81% de previsão de pessoas que morreram e 46% de pessoas que sobreviveram, novamente um valor abaixo de 50% na assertividade.

O modelo que apresentou melhor resultado foi o Decision Tree. Como o nome indica, esse algoritmo pode ser utilizado para predizer valores numéricos. Basicamente, nesse caso, a divisão consiste em encontrar grupos com resultados similares, e a média dos seus resultados é usada como predição para esse grupo. O algoritmo encontra a melhor divisão tentando minimizar a variância do grupo. O modelo teve assertividade na previsão de passageiros mortos de 75%, já para sobreviventes 53% o único valor acima de 50%.

Dessa forma, percebemos que dos três modelos testados, Decision Tree teve melhor performance, portanto é possível afirmar que esse modelo é o mais recomendado para predição de sobrevivência ou mortes de passageiros e tripulantes do Navio Titanic.

7. Links

Conforme solicitado, seguem abaixo os links para acesso ao vídeo de apresentação e para o repositório contendo o notebook em Python para elaboração das análises e modelos utilizados no projeto.

Link para o Git:

https://github.com/gabrielcouto2020/DataScienceBigDataTCC

Link para o vídeo:

https://www.loom.com/share/60a0af5150964a01862a26c7c104d633

REFERÊNCIAS

Vasandani, Jasmine**. A Data Science Workflow Canvas to Kickstart Your Projects**. Disponível em: https://towardsdatascience.com/a-data-science-workflow-canvas. Acesso em: 01/2022

Kaggle. **Titanic - Machine Learning from Disaster** - Disponível em: https://www.kaggle.com/competitions/titanic/overview/description. Acesso em 10/2021

Fake Name Generator. Disponível em: https://pt.fakenamegenerator.com. Acesso em: 10/2021

Harrison, Matt. **Machine Learning Pocket Reference: Working with Structured Data in Python**. O Reilly, 2019

Data Science Academy. **Python Fundamentos Para Análise de Dados 3.0** - Disponível em: https://www.datascienceacademy.com.br/cursosgratuitos. Acesso em 09/2021

Viana, Suzana. **O que aprendemos com o Titanic? Uma análise de Data Science.** Disponível em: https://suzana-svm.medium.com/data-science-udacity-titanic-e5b04a8e415f. Acesso em: 12/2021

Melo, Carlos. **Data Science: Investigando o naufrágio do Titanic**. Disponível em: https://sigmoidal.ai/data-science-titanic-python-2. Acesso em 12/2021

Nussbaumer, Cole. **Storytelling com Dados, um guia sobre visualização para profissionais de dados**. Rio de Janeiro, Alto Books, 2018

APÊNDICE

Exploração de aprendizado de máquina

O banco de dados foi enriquecido com dados sobre as pessoas!

 # Exploração de aprendizado de máquina

****Esse notebook tem como objetivo mostrar a exploração de dados sobre o Titanic, o processo inclui**:**

- Coleta de dados

**- Entendendo erros e tratando dataset **

**- Tratando e explorando novas features **

- Explorando o conjunto de dados

**- Desenvolvimento de modelos de Machine Learning **

Coleta de dados<a name='Coleta de dados' />

Importando dados

```python

import pandas as pd

import numpy as np

import zipfile

import requests

from io import BytesIO

import os

import seaborn as sns

```
import matplotlib.pyplot as plt

from sklearn.model_selection import KFold, cross_val_predict, cross_val_score
from sklearn.svm import SVC
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn import metrics
from sklearn.model_selection import ShuffleSplit
from IPython.core.pylabtools import figsize

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier


df = pd.read_csv('/Users/gabrielfelipepinheirocouto/Documents/arquivos_machine_learning_tcc/data/database_one.csv')
dfo = pd.read_csv('/Users/gabrielfelipepinheirocouto/Documents/arquivos_machine_learning_tcc/data/database_two.csv')
```

```python
df.head(3)
```

```python
dfo.head(3)
```

```

```python
len(df)
len(dfo)
```

Explorando tipos de dados

```python
df.dtypes
```

```python
dfo.dtypes
```

Desenvolvendo uma função, a fim de ajustar as colunas e torná-las padronizadas

```python
import re

def correct_columns(col_name):
   return re.sub(r"[/| ]", "", col_name).lower()
```

Chamando função para ajuste nas colunas
```

```python
df.columns = [correct_columns(col) for col in df.columns]
```

```python
df.head(3)
```

```python
dfo.columns = [correct_columns(col) for col in dfo.columns]
```

```python
dfo.head(3)
```

Validaçao e join entre os datasets

```python
len(dfo)
```

```python
len(df)
```

```python
union = df.join(dfo.set_index('number'), on='number', how='inner')
df = union
```

```python
df.head(3)
```

Entendendo erros e tratando dataset <a name='Entendendo erros e tratando dataset' />

Explorando na colunas do dataset

```python
df.head(4).isnull
```

```python
for field in df.columns:
   print(field, 'NaN:', df[field].isnull().sum())
```

Tratando os NaN

```python
for dataset in [df]:
```

dataset.loc[dataset['embarked'].isnull(), 'embarked'] = 0

```

Entendendo que os dados de cabine não são relavante para nosso modelo, portando iremos eliminar

```python

df = df.drop(['cabin'], axis=1)

```

Como temos apenas o 2 sexos em nossa base, iremos ajustar para 1 e o outro 0

```python

df['sex'] = df['sex'].map({'female': 1, 'male': 0}).astype(int)

```

Entendendo os dados de pessoas com erros na idade

```python

df.loc[df['age'].isnull()][(df['sibsp'] > 0) & (df['parch'] > 0)]

```

Iremos pegar a média de idade de passageiros, com caracteristicas similares a quem está com NaN em idade, tudo isso para criação do nosso For

```python

guess_ages = np.zeros((2,3))

```
for dataset in [df]:
   for sex in range(0, 2):
      for pclass in range(0, 3):
         guess_df = dataset[(dataset['sex'] == sex) & (dataset['pclass'] == pclass + 1)]['age'].dropna()

         age_guess = guess_df.median()
         print('sex', sex, 'pclass', pclass)
         print(len(guess_df), age_guess)

         # Converte para a idade mais proxima (para nao ficar 0.4 ou 0.333)
         guess_ages[sex, pclass] = int( age_guess / 0.5 + 0.5 ) * 0.5

   for sex in range(0, 2):
      for pclass in range(0, 3):
         dataset.loc[ (dataset.age.isnull()) & (dataset.sex == sex) & (dataset.pclass == pclass + 1), 'age'] = guess_ages[sex,pclass]

   dataset['age'] = dataset['age'].astype(int)
```

Validando se temos alguma linha NaN

```python
df.loc[df['age'].isnull()]
```

```python
print('df:', len(df.loc[df['age'].isnull()]))
```

```python
for field in df.columns:
** print(field, 'NaN:', df[field].isnull().sum())**
```

Notamos que as colunas statefull e company tem linhas NaN, como se trata de uma linha, iremos exclui-las da base

```python
len(df)
```

```python
df = df.dropna()
```

```python
len(df)
```

```python
for field in df.columns:
** print(field, 'NaN:', df[field].isnull().sum())**
```

```python
df
```

Tratando e explorando novas features <a name='Tratando e explorando novas features' />

Tratando e explorando novas features, iremos transformar dados categóricos em números

```python
pd.crosstab(df["title"], df['sex'])
```

Transformando a coluna sex em números

```python
title_mapping = {"Dr.": 1, "Mr.": 2, "Mrs.": 3, "Ms.": 4}

for dataset in [df]:
   dataset['title'] = dataset['title'].map(title_mapping)
   dataset['title'] = dataset['title'].fillna(0)
```

```python
pd.crosstab(df['title'], df['sex'])
```

```python
df.head(3)
```

Aqui separamos casos que a pessoa está sozinho ou não, e o tamanho da famila

```python
df['familysize'] = df['sibsp'] + df['parch'] + 1

df['Isalone'] = 0

df.loc[df['familysize'] == 1, 'Isalone'] = 1
```

```python
df.head(3)
```

Explorando mais dados para ver sua relevância no modelo, e exclusão de colunas sem necessidade

```python
np.unique(df["nameset"])
```

```python
df = df.drop(['nameset'], axis=1)
```

```

```python
np.unique(df["telephonenumber"])
```

```python
df = df.drop(['telephonenumber'], axis=1)
```

```python
np.unique(df["tropicalzodiac"])
```

```python
df = df.drop(['tropicalzodiac'], axis=1)
```

```python
df.head(3)
```

```python
np.unique(df["fulcolor"])
```
```

```python
title_mapping = {"black": 1, "dark skinned": 2, "white": 3}

for dataset in [df]:
   dataset['fulcolor'] = dataset['fulcolor'].map(title_mapping)
   dataset['fulcolor'] = dataset['fulcolor'].fillna("")
```

```python
np.unique(df["fulcolor"])
```

```python
df.head(3)
```

Observamos mais outros dados que não precisaremos, portanto iremos excluilos e deixar nosso dados mais aderentes ao modelo

```python
df = df.drop(['ticket',"state","statefull","ccnumber","occupation","company","city","streetaddress","name", "centimeters","kilograms","number","title"], axis=1)
```

```python
df.head(3)
```

```

```python
df.info()
```

```python
df['fare'].describe()
```

Ja excluimos as colunas que não devem fazer parte do nosso modelo, agora vamos continuar a trasformação de dados

```python
for dataset in [df]:
    dataset['fareCat'] = 0
    dataset.loc[dataset.fare < 7.91, 'fareCat'] = 1
    dataset.loc[(dataset.fare >= 7.91) & (dataset.fare <= 14.45), 'fareCat'] = 2
    dataset.loc[(dataset.fare >= 14.45) & (dataset.fare <= 31), 'fareCat'] = 3
    dataset.loc[dataset.fare > 31, 'fareCat'] = 4

df.head(3)
```

```python
df.embarked.unique()
```
```

```python
embarked_mapping = {"S": 1, "C": 2, "Q": 3, 0: 0}
for dataset in [df]:
    dataset['embarked'] = dataset['embarked'].map(embarked_mapping)

df.head(3)
```

Explorando o conjunto de dados <a name='Explorando o conjunto de dados' />

explorando e analisando os dados

```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style ('whitegrid')
%matplotlib inline
```

Percentual de sobreviventes

```python
print('Survival percentage:', len(df[df.survived == 1]) / len(df))
sns.countplot(x="survived", data=df)
```

Porcentagem de sobrevivência de pessoas que estavam na primeira classe

```python
print('Survival percentage:', len(df[(df.pclass == 1) & (df.survived == 1 )]) / len(df))
sns.countplot(x="survived", data=df[df.pclass == 1])

```

Porcentagem de sobrevivência de pessoas que estavam na segunda classe

```python
print('Survival percentage:', len(df[(df.pclass == 2) & (df.survived == 1 )]) / len(df))
sns.countplot(x="survived", data=df[df.pclass == 2])
```

Porcentagem de sobrevivência de pessoas que estavam na terceira classe

```python
print('Survival percentage:', len(df[(df.pclass == 3) & (df.survived == 1 )]) / len(df))
sns.countplot(x="survived", data=df[df.pclass == 3])
```

Entendendo se mulheres de primeira classe sobreviveram

```python
print(len(df[(df.pclass == 1) & (df.sex == 1 ) & (df.survived == 1)]) / len(df))
sns.countplot(x="survived", data=df[(df.pclass == 1) & (df.sex == 1 )])
```

Entendendo por que algumas das mulheres de primeira classe morreram

```python
df[(df.sex == 1) & (df.survived == 0) & (df.pclass == 1)]
```

Matrix de correlação

```python
df.head(3)
```

```python
pt = df.drop(['sibsp',"parch","embarked","fulcolor"], axis=1)
```

```python
sns.heatmap(pt.corr(), annot=True, vmin=-1, vmax=1);
```

```python
df
```

Desenvolvimento de modelos de Machine Learning <a name='Desenvolvimento de modelos de Machine Learning' />

Explorando os dados já tratados e prontos para serem usados e separando nossa base de teste e treinamento

```python
msk = np.random.rand(len(df)) < 0.7
train = df[msk]
test = df[~msk]
```

```python
X_train = train.drop(["survived"], axis=1)
Y_train = train["survived"]
X_test = test.drop("survived", axis=1)
Y_test = test["survived"]
```

Modelos de Machine Learning

```python
modelos = [LogisticRegression(solver='liblinear'), RandomForestClassifier(n_estimators=400, random_state=42),
DecisionTreeClassifier(random_state=42)]
```

Validação cruzada

```python
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

```

```python
mean=[]
std=[]
for model in modelos:
  result = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy', n_jobs=-1)
  mean.append(result)
  std.append(result)
```

```python
classificadores = ['Logistic Regression', 'Random Forest', 'Decision Tree']

plt.figure(figsize=(20, 10))
for i in range(len(mean)):
  sns.distplot(mean[i], hist=False, kde_kws={"shade":True})

plt.title("Distribuição de cada um dos modelos", fontsize=15)
plt.legend(classificadores)
plt.xlabel("Acuracia", labelpad=20)
plt.yticks([])
```

```python
rf_clf = RandomForestClassifier(n_estimators=400, n_jobs=-1, random_state=42)
dt_clf = DecisionTreeClassifier(random_state=42)
lr_clf = LogisticRegression(solver='liblinear')
```
```

```
rf_clf.fit(X_train, Y_train)
dt_clf.fit(X_train, Y_train)
lr_clf.fit(X_train, Y_train)
```

```python
rf_prob = rf_clf.predict_proba(X_test)
dt_prob = dt_clf.predict_proba(X_test)
lr_prob = lr_clf.predict_proba(X_test)

rf_preds = np.argmax(rf_prob, axis=1)
dt_preds = np.argmax(dt_prob, axis=1)
lr_preds = np.argmax(lr_prob, axis=1)
```

```python
cm1 = metrics.confusion_matrix(Y_test, rf_preds)
cm1 = cm1.astype('float') / cm1.sum(axis=1)[:, np.newaxis]

cm2 = metrics.confusion_matrix(Y_test, dt_preds)
cm2 = cm2.astype('float') / cm2.sum(axis=1)[:, np.newaxis]

cm3 = metrics.confusion_matrix(Y_test, lr_preds)
cm3 = cm3.astype('float') / cm3.sum(axis=1)[:, np.newaxis]
```

```python
classes=["Died", "Alive"]
f, ax = plt.subplots(1, 3,  figsize=(30, 5))
```

```
ax[0].set_title("Random Forest", fontsize=15.)
sns.heatmap(pd.DataFrame(cm1, index=classes, columns=classes),
        cmap='winter', annot=True, fmt='.2f', ax=ax[0]).set(xlabel="previsão", ylabel="Valor
real")

ax[1].set_title("Decision tree", fontsize=15.)
sns.heatmap(pd.DataFrame(cm2, index=classes, columns=classes),
         annot=True, fmt='.2f', ax=ax[1]).set(xlabel="previsão", ylabel="Valor real")

ax[2].set_title("LogisticRegression", fontsize=15.)
sns.heatmap(pd.DataFrame(cm3, index=classes, columns=classes),
        cmap='Greens', annot=True, fmt='.2f', ax=ax[2]).set(xlabel="previsão", ylabel="Valor
real")
```